树、二叉树、遍历

2019年2月9日 21:43

-
- 树

1. 树（Tree）是由n（n > 0）个结点组成的有限集合T， 满足
    1) 有且仅有一个特定的称为该树的根（Root）的结点
    2) 除根结点之外的其余结点可分为m(m > 0)个互不相交的集合T1，T2，…，Tm，且其中每个集合又是一棵树，并称之为根的子树(Subtree)数据元素之间的逻辑结构关系
2. 树的结点：包含一个数据元素及若干个指向其子树的分支
3. 结点的度和树的度
    1) 结点的度：结点的子树的个数
    2) 树的度：树中结点度的最大值
4. 叶子和分支结点
    1) 终端结点/叶子：度为零的结点
    2) 分支结点/非终端结点：度不为零的结点
5. 孩子、双亲及兄弟结点
    1) 某结点的各子树的根称为该结点的孩子，而该结点称为孩子的双亲
    2) 具有相同双亲的结点互称为兄弟
    3) 一棵树上除根结点以外的其他结点称为根的子孙，而根结点是其子孙的祖先
6. 结点的层次和树的深度
    1) 根的层次值为1
    2) 结点的层次值从根算起，非根结点的层次值为双亲结点层次值加1
    3) 树中结点的最大层次值称为树的深度或高度
7. 有序树和无序树
    1) 如果树中结点的各子树从左至右是有次序的（即不能互换），则称该树为有序树，否则称该树为无序树。在有序树中最左边的子树的根称为该树的第一个孩子，最右边的子树的根称为该树的最后一个孩子
8. 森林：m（m > 0）棵互不相交的树的集合
    1) 对树中每个结点而言，其子树的集合即为森林

-
- 二叉树

1. 定义：N(N≥0)个结点的有限集合。它或为空树 (N=0)、或由一个根结点和两个分别称为左子树和右子树的互不相交的二叉树构成
    1) 满二叉树full binary tree：深度为k并且含有2^(k-1)个结点的二叉树
    2) 满二叉树可以从根结点开始自上向下，自左至右顺序编号
    3) 完全二叉树complete binary tree：深度为k，含有n个结点的二叉树，每个结点的编号与相应满二叉树结点顺序编号从1到n相对应的二叉树
    4)
2. 性质

1) 第i(i≥1)层上至多有2^(i-1)个结点（数学归纳法证明）
2) 深度为k(k≥1)的二叉树至多有2^k-1个结点（性质一求和）
3) 设叶个数为n0，度为1的结点数为n1，度为2的结点个数为n2，则n0 = n2+1
   i. 结点总数n = n0 +n1+n2
   ii. 由于有n个结点的二叉树总分枝数为n-1条，于是n-1 = 0*n0 +1*n1+2*n2
   iii. 代入化简，得n0 =n2+1n0 = n2+1
4) 具有n个结点的完全二叉树，其深度为 [log2n] +1 ([x]表示向下取整)
5) 若对有n个结点的完全二叉树进行顺序编号(1≤i≤n)， 对编号为i(i≥1)的结点,有
   i. 当i=1时，该结点为根，它无双亲结点
   ii. 当i > 1时，该结点的双亲结点编号为 i / 2
   iii. 若2i≤n，则有编号为2i的左孩子，否则没有左孩子
   iv. 若2i+1≤n，则有编号为2i+1的右孩子，否则没有右孩子
   -
   - 二叉树的存储结构

1. 顺序存储结构
   1) 根据二叉树性质5，结点在一维数组中的相对位置隐含着结点之间的关系。因此在数组bt中可以方便地由某结点bt[i]的下标i找到它们的双亲结点 bt[i/2]，或左、右孩子结点bt[2i]、 bt[2i+1]
2. 链式存储结构
   1) 最常用的是二叉链表和三叉链表
   2) 二叉链表的每个结点有一个数据域和两个指针域，一个指针指向左孩子，另一个指向右孩子。结点结构描述为：
      i.
```
typedef struct btnode{
        ElemType data;   /* 数据域*/
        struct btnode *lchild, *rchild;    /* 左、右孩子指针域*/
}BTNode;
```
   3) 建立二叉链表结构的二叉树
      i.
```
#define MAXNUM 14 /*结点的最大编号*/
typedef char ElemType; /* 数据类型*/
typedef struct btnode {
        ElemType data; /* 数据域*/
        struct btnode *lchild, *rchild; /* 左、右孩子指针域*/
}BTNode;
BTNode *p[MAXUNM + 1]; /* 辅助向量*/
BTNode *Creat_Bt(void) { /* 建立二叉链表 */
        printf("\nenter (i,ch) until enter(0,#):\n"); /*输入数对(0,#)时结束*/
        scanf("%d,%c", &i, &ch);
        while (i != 0 && ch != '#') {
                s = (BTNode*)malloc(sizeof(BTNode));
                s->data = ch;
                s->lchild = s->rchild = NULL;
                p[i] = s;
                if (i == 1)
                        t = s;
```

```
        else{
            j = i / 2;
            if (i % 2 == 0)
                p[j]->lchild = s;
            else
                p[j]->rchild = s; }
        printf("\n enter i,ch:");
        scanf("%d,%c", &i, &ch); }
    return t; } /* Creat_Bt */
```

-
- 遍历二叉树

3. 先根遍历二叉树
   1) 递归定义为：若二叉树非空，则：(1) 访问根结点；(2) 按先根次序遍历左子树；(3) 按先根次序遍历右子树；否则，遍历结束
   2)
```
void Preorder(BTNode *bt) {
    if (bt) {
        printf(bt->data); /*访问根结点*/
        Preorder(bt->lchild); /*先根遍历左子树*/
        Preorder(bt->rchild); /*先根遍历右子树*/ } } /* Preorder */
```
   3) 可以用栈把递归算法写成等价非递归算法
      i. 假定是二叉树有n个结点，由于每个结点仅被访问一次，每个结点的指针要进一次栈，出一次栈，因此，算法中的输出语句、进栈和出栈的操作均被执行n次，算法的时间复杂度为O(n)
      ii. 有n个结点的树的深度最大值为n， 因此栈所需要的最大容量不超过n
   4)
```
void Preorder2(BTNode *bt) {
/* 先序遍历二叉树bt非递归算法 ， S是BTNode *类型的栈 */
    p = bt;
    InitStack(S); /*置栈空*/
    while (p || !StactEmpty(S)) {
        if (p) { /* 二叉树非空 */
            printf(p->data); /*访问根结点*/
            Push(S, p); /*根指针进栈*/
            p = p->lchild; /*p移向左孩子*/ }
        else { /*栈非空*/
            Pop(S, p); /*双亲结点出栈*/
            p=p->rchild; /*p移向右孩子*/ }
        } /* 二叉树空且栈空*/ } /*Preorder2 */
```

4. 求叶结点
   1) 结点bt为叶子的条件：bt->lchild==NULL && bt->rchild==NULL
   2) 例：用先根遍历对每个结点访问一次，顺便计数
   3)
```
int count = 0;
void Leafs(BTNode *bt) {
    if (bt) {
        if (bt->lchild == NULL && bt->rchild == NULL) {
            printf("%3c", bt->data);
            count++; }
```

```
        Leafs(bt->lchild);
        Leafs(bt->rchild); } } /*Leafs*/
```

5. 中根遍历二叉树
   1) 递归定义：若二叉树非空，则：(1) 按中根次序遍历左子树；(2) 访问根结点；(3) 按中根次序遍历右子树；否则，遍历结束
   2)
```
void Inorder(BTNode *bt) {
    if (bt) {
        Inorder(bt->lchild); /*中根遍历左子树*/
        printf(bt->data); /* 访问根结点 */
        Inorder(bt->rchild); /* 中根遍历右子树*/ } } /*Inorder*/
```
   3)
```
void Inorder2(BTNode *bt) {
/*中序遍历二叉树bt的非递归算法，S是BTNode *类型的栈 */
    p = bt; InitStact(S); /*置栈空*/
    while (p || !StactEmpty(S)) {
        if (p) {
            Push(S, p);
            p = p->lchild; }
        else{
            Pop(S, p);
            printf(p->data); /* 访问根结点*/
            p = p->rchild; } } } /* Inorder2 */
```

6. 按层次遍历二叉树（队列实现）
   1)
```
typedef struct tnode {
    elemtype data;
    struct tnode *lchild, *rchild;
}tnode, *bintree;
void leveltree(bintree t) {
    bintree p,  q[max];
    int front , rear;
    front = rear = 0;
    rear = (rear+1)% max;
    q[rear]=t;
    while (front!=rear) {
        front=(front+1)%max;
        p = q[front];
        printf("%c", p->data);
        if (p->lchild) {
            rear=(rear+1)%max;
            q[rear]=p->lchild; }
        if (p->rchild) {
            rear=(rear+1)%max;
            q[rear]=p->rchild; } } }
```

7. 打印逆时针旋转90°的二叉树（凹入表示法打印）
   1) 由于把二叉树逆时针旋转90º后，在屏幕上先显示的是右子树，然后是根结点，最后是左子树，所以可以利用二叉树的中根遍历原理，将算法作调整，即“中根遍历右子树，访问根结点，中根遍历左子树”。访问操作为输出结点的数据域，输出位置要根据结点所在的层次进行调整
   2)
```
void PrintTree(BTNode *bt, int level) {
    int i;
    if (bt) {
```

```
        PrintTree(bt->rchild, level + 1); /*右子树level+1层结点横向显示 */
        if (level != 1) {
            for (i = 1; i < 6 *(level - 1); i++)
            printf(" "); /*输出空格*/
            printf("----%c\n", bt->data); }
        else
            printf(" %c\n", bt->data);
        PrintTree(bt->lchild, level + 1);/*左子树level+1层结点横向显示 */
    } } /*PrintTree*/
```

8. 后根遍历二叉树
   1) 递归定义：若二叉树非空，则：(1) 按后根次序遍历左子树；(2) 按后根次序遍历右子树；(3)访问根结点； 否则，遍历结束
   2)
```
void Postorder(BTNode *bt) {
/* 后序遍历二叉树bt的递归算法 */
    if (bt) {
        Postorder(bt->lchild);
        Postorder(bt->rchild);
        printf(bt->data); } } /*Postorder*/
```

9. 销毁二叉树：在释放某个结点的存储空间前必须先释放其左右孩子结点的存储空间。可以用后根遍历来实现
   1)
```
void DestroyTree(BTNode *bt) { /*销毁二叉树bt*/
    if (bt && bt->lchild)
        DestroyTree(bt->lchild); /*销毁左子树*/
    if (bt && bt->rchild)
        DestroyTree(bt->rchild); /*销毁右子树*/
    printf("%3c", bt->data); /* 测试语句*/
    free(bt); /*释放根结点的存储空间 */} /*DestroyTree*/
```

      i.
      ii.
      iii.
      iv.
      v. --------我是底线----------